Computer Science 271
Project 0100
Due Wednesday, March 1

You will complete this project with a partner. Both individuals are expected to contribute equally to all parts of the project.

1. Implement a minimum priority queue ADT as a template class in C++. The template class should *inherit* from your `MinHeap` template class. Here is the template class definition.

```
template <class KeyType>
class MinPriorityQueue : public MinHeap<KeyType>
{
  public:
    MinPriorityQueue();             // default constructor
    MinPriorityQueue(int n);        // construct an empty MPQ with capacity n
    MinPriorityQueue(const MinPriorityQueue<KeyType>& pq);  // copy constructor
    // Destructor inherited from MinHeap<KeyType>

    KeyType* minimum() const;                    // return the minimum element
    KeyType* extractMin();                       // delete the minimum element and return it
    void decreaseKey(int index, KeyType* key);  // decrease the value of an element
    void insert(KeyType* key);                   // insert a new element
    bool empty() const;                          // return whether the MPQ is empty
    int length() const;                          // return the number of keys
    std::string toString() const;                // return a string representation of the MPQ

    // Assignment operator inherited from MinHeap<KeyType>

    // Specify that MPQ will be referring to the following members of MinHeap<KeyType>.

    using MinHeap<KeyType>::A;
    using MinHeap<KeyType>::heapSize;
    using MinHeap<KeyType>::capacity;
    using MinHeap<KeyType>::parent;
    using MinHeap<KeyType>::swap;
    using MinHeap<KeyType>::heapify;

    /* The using statements are necessary to resolve ambiguity because
       these members do not refer to KeyType.  Alternatively, you could
       use this->heapify(0) or MinHeap<KeyType>::heapify(0).
    */
};

template <class KeyType>
std::ostream& operator<<(std::ostream& stream, const MinPriorityQueue<KeyType>& pq);

class FullError { };    // MinPriorityQueue full exception
class EmptyError { };   // MinPriorityQueue empty exception
class KeyError { };     // MinPriorityQueue key exception
```

Notes:

- For the `MinPriorityQueue` class to access the private instance variables of the `MinHeap` class, you will need to change the `private` members of `MinHeap` to be `protected` instead.
- To work with the application you will implement next, the `MinPriorityQueue` must contain an array of *pointers* to items rather than the items themselves. This requires the following changes:
  - The instance variable `A` in `MinHeap` must be declared as `KeyType **A` and initialized as

    ```
    A = new KeyType*[capacity];
    ```

    in each constructor.
  - The `heapSort` method and the constructor that takes in an array should now have parameters that are arrays of `KeyType*`.
  - Whenever you compare two keys in `heapify`, you will need to dereference each pointer to an item. For example, you will compare the key of the parent to the key of its left child with `*(A[left]) < *(A[index])`.
  - In your `toString` method, you will also need to dereference the keys that you "print."
  - The type of the `temp` variable in the `swap` method will need to change to `KeyType*`.
- The generic class `KeyType` is the type of the data contained in the priority queue. We assume that `KeyType` has overloaded the `<` relational operator and the `<<` stream operator. The `<` operator *needs to compare the keys* inside the items of class `KeyType`. In this way, the `KeyType` class can contain both a key and "satellite data," and we do not have to explicitly specify the type of the key values.
- Include suitable preconditions and postconditions in the comments before each method.
- Your methods should throw appropriate exceptions when the parameters do not satisfy preconditions.
- Include unit tests (using `assert` and the `toString` method) for each of your methods.

2. Write a program that can compress a text file using Huffman coding and decompress a file that was previously compressed (by your program).

   - Your program should accept command-line parameters (using `argc` and `argv`) telling it whether to compress or decompress. If the first command line parameter is `-c`, then the next two command line parameters indicate the source and destination file names, respectively. For example,

     ```
     huffman -c foo.txt foo.huff
     ```

     should compress the file `foo.txt` into the output file `foo.huff`. On the other hand, if the first command-line parameter is `-d`, then the next two command line parameters indicate the compressed and destination file names, respectively. For example,

     ```
     huffman -d foo.huff foo.txt
     ```

     should decompress the file `foo.huff` into the text file `foo.txt`.
   - For partial credit, your compressed file may consist of `0` and `1` characters instead of bits. For full credit, your compressed file must really be compressed, i.e., encode characters at the bit level.

- You will need to use your MinPriorityQueue template class in your implementation. An element in your min-priority queue will be a node in the Huffman tree. Each node will need to contain a character and a frequency. As explained above, the < operator and << stream operator must be overloaded for your node class.
- I recommend that you tackle this in stages:
  (a) Implement compression first, writing 0 and 1 characters to the output file.
  (b) Devise a scheme to efficiently store the code at the beginning of a compressed file.
  (c) Implement decompression.
  (d) Improve your program by having it read and write bit strings instead of 0 and 1 characters.

Please submit via Notebowl your pq.cpp, test_pq.cpp, node.h, and huffman.cpp source files, and a PDF named proj4_yourname.pdf containing these source files (using enscript). Just submit one submission per pair. Be sure to indicate the names of both group members on all of your submitted files.